

# Scikit-learn the Python module

Jens Lundström

2013-03-20

## References:

F.Pedregosa, G. Varoquaux, A. Gramfort, V. Michel and B.Thirion, Scikit-learn: *Machine Learning in Python*, Journal of Machine Learning Research 12 (2011) 2825-2830

<http://scikit-learn.org>

<http://www.raspberrypi.org/>

<http://docs.cython.org/src/userguide/tutorial.html>

<https://developers.google.com/open-source/soc/>



# What is Scikit-learn?

- A Python module providing algorithms for supervised and unsupervised machine learning.
- Directed to *non-specialists*.
- Documentation: "*we try to minimize the use of machine-learning jargon*".
- Focus areas: Ease of use, performance, documentation and API consistency.
- Originated from a *Google Summer of Code* in 2007. Started by David Cournapeau. 32 people mentioned at [scikit-learn.org](http://scikit-learn.org).

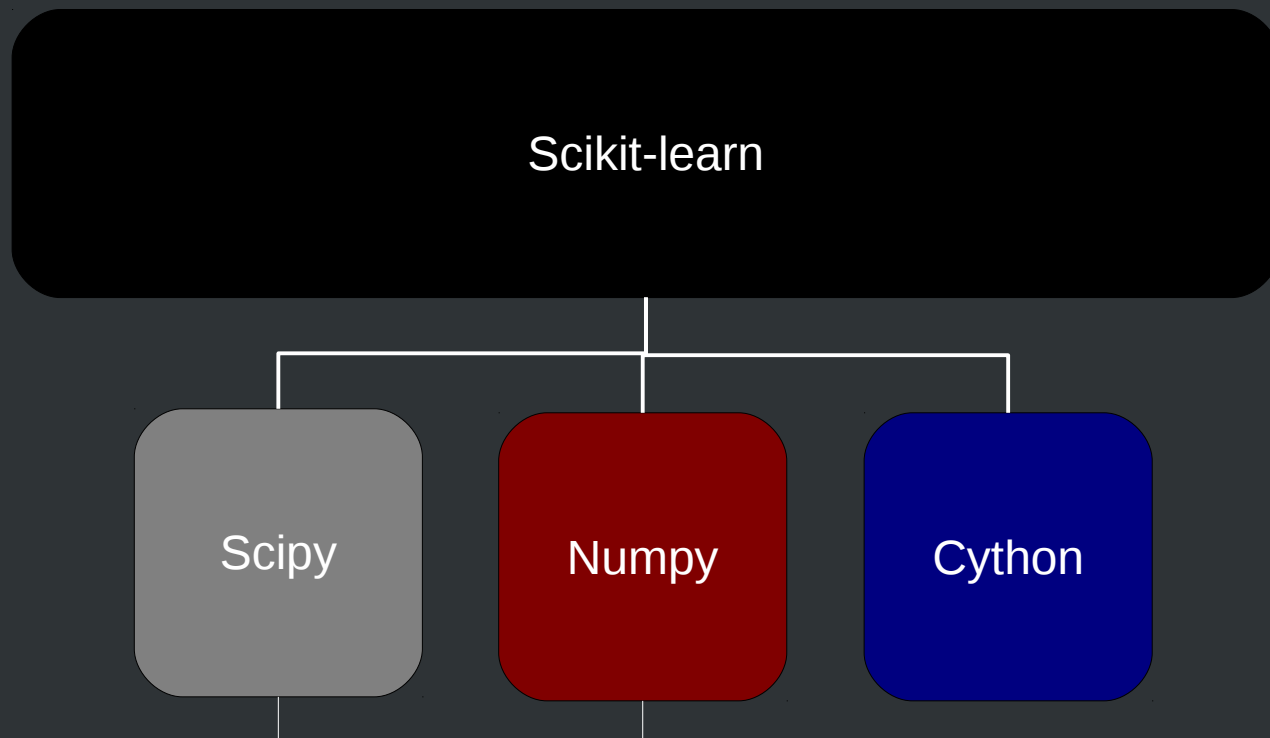


# WAIT! What is: Google Summer of Code

- Program (obviously by Google) which provides stipends to students doing a open-source project during the summer.
- At least 300 projects (2005-2012)
  - Example: Widgets for statistics – Orange DM
- Google code-in (pre-university)



# Dependencies



# WAIT! What is: Cython

- C-extensions for Python
  - *Superset of the Python language that additionally supports calling C functions and declaring C types on variables and class attributes.*

- Example:

save as primes.pyx

compile using

```
$ python setup.py build_ext --inplace
```

use

```
>>> import primes
>>> primes.primes(10)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

```
1  def primes(int kmax):
2      cdef int n, k, i
3      cdef int p[1000]
4      result = []
5      if kmax > 1000:
6          kmax = 1000
7      k = 0
8      n = 2
9      while k < kmax:
10         i = 0
11         while i < k and n % p[i] != 0:
12             i = i + 1
13         if i == k:
14             p[k] = n
15             k = k + 1
16             result.append(n)
17         n = n + 1
18     return result
```



# What is Scikit-learn really?

- Simply: collection of Python files implementing various ML algorithms. From /sklearn:

```
j1un@ubuntu: /usr/local/lib/python2.7/dist-packages/sklearn
j1un@ubuntu:/usr/local/lib/python2.7/dist-packages/sklearn$ ls
base.py          _hmmc.so        pipeline.py
base.pyc         hmm.py          pipeline.pyc
_build_utils.py  hmm.pyc         pls.py
_build_utils.pyc __init__.py     pls.pyc
__check_build    __init__.pyc    preprocessing.py
cluster          kernel_approximation.py preprocessing.pyc
covariance       kernel_approximation.pyc qda.py
cross_validation.py lda.py          qda.pyc
cross_validation.pyc lda.pyc        semi_supervised
datasets         linear_model    setup.py
decomposition    manifold        setup.pyc
ensemble         metrics         svm
externals        mixture         tests
feature_extraction multiclass.py   test_setup.py
feature_selection multiclass.pyc  test_setup.pyc
gaussian_process naive_bayes.py  tree
grid_search.py   naive_bayes.pyc utils
grid_search.pyc  neighbors
j1un@ubuntu:/usr/local/lib/python2.7/dist-packages/sklearn$
```



# What is Scikit-learn really?

- Example

matplotlib! →

Simple to load (toy)  
datasets! →

Create an instance of  
a classifier (SVM) →

Fit the model  
according to the  
training data + make  
predictions of the test  
data →

```
print __doc__

import random
import pylab as pl
from sklearn import svm, datasets
from sklearn.metrics import confusion_matrix

# import some data to play with
iris = datasets.load_iris()
X = iris.data
y = iris.target
n_samples, n_features = X.shape
p = range(n_samples)
random.seed(0)
random.shuffle(p)
X, y = X[p], y[p]
half = int(n_samples / 2)

# Run classifier
classifier = svm.SVC(kernel='linear')
y_ = classifier.fit(X[:half], y[:half]).predict(X[half:])

# Compute confusion matrix
cm = confusion_matrix(y[half:], y_)

print cm

# Show confusion matrix
pl.matshow(cm)
pl.title('Confusion matrix')
pl.colorbar()
pl.show()
```



# sklearn.datasets

```
iris = datasets.load_iris()
```

return Bunch!

\* show code `print(iris)`

```
jlu@ubuntu: /usr/local/lib/python2.7/dist-packages/sklearn/datasets
GNU nano 2.2.6 File: base.py

>>> data = load_iris()
>>> data.target[[10, 25, 50]]
array([0, 0, 1])
>>> list(data.target_names)
['setosa', 'versicolor', 'virginica']
"""
module_path = dirname(__file__)
data_file = csv.reader(open(join(module_path, 'data', 'iris.csv')))
fdescr = open(join(module_path, 'descr', 'iris.rst'))
temp = next(data_file)
n_samples = int(temp[0])
n_features = int(temp[1])
target_names = np.array(temp[2:])
data = np.empty((n_samples, n_features))
target = np.empty((n_samples,), dtype=np.int)

for i, ir in enumerate(data_file):
    data[i] = np.asarray(ir[:-1], dtype=np.float)
    target[i] = np.asarray(ir[-1], dtype=np.int)

return Bunch(data=data, target=target,
             target_names=target_names,
             DESCR=fdescr.read(),
             feature_names=['sepal length (cm)', 'sepal width (cm)',
                           'petal length (cm)', 'petal width (cm)'])

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```





# sklearn.datasets

The Bunch class

```
jlun@ubuntu: /usr/local/lib/python2.7/dist-packages/sklearn/d
GNU nano 2.2.6 File: base.py

import shutil
from os import environ
from os.path import dirname
from os.path import join
from os.path import exists
from os.path import expanduser
from os.path import isdir
from os import listdir
from os import makedirs

import numpy as np

from ..utils import check_random_state

class Bunch(dict):
    """Container object for datasets: dictionary-like object that
    exposes its keys as attributes."""

    def __init__(self, **kwargs):
        dict.__init__(self, kwargs)
        self.__dict__ = self

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```



# Doing the same thing (online)

```
jlu@ubuntu: /usr/local/lib/python2.7/dist-packages/sklearn/d
GNU nano 2.2.6 File: mldata.py

Returns
-----

data : Bunch
    Dictionary-like object, the interesting attributes are:
    'data', the data to learn, 'target', the classification labels,
    'DESCR', the full description of the dataset, and
    'COL_NAMES', the original names of the dataset columns.

Examples
-----
Load the 'iris' dataset from mldata.org:
>>> from sklearn.datasets.mldata import fetch_mldata
>>> iris = fetch_mldata('iris')
>>> iris.target[0]
1
>>> print(iris.data[0])
[-0.555556  0.25      -0.864407 -0.916667]

Load the 'leukemia' dataset from mldata.org, which needs to be transposed
to respects the sklearn axes convention:
>>> leuk = fetch_mldata('leukemia', transpose_data=True)
>>> print(leuk.data.shape[0])

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```



# How about

```
classifier = svm.SVC(kernel='linear')
y_ = classifier.fit(X[:half], y[:half]).predict(X[half:])
```

?

- Legacies

```
jlun@ubuntu: /usr/local/lib/python2.7/dist-packages/sklearn/s
GNU nano 2.2.6 File: classes.py

class SVC(BaseSVC):
    """C-Support Vector Classification.

    ^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
    ^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

```
jlun@ubuntu: /usr/local/lib/python2.7/dist-packages/sklearn/s
GNU nano 2.2.6 File: base.py

class BaseSVC(BaseLibSVM, ClassifierMixin):
    """ABC for LibSVM-based classifiers."""

    ^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
    ^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

```
jlun@ubuntu: /usr/local/lib/python2.7/dist-packages/sklearn/s
GNU nano 2.2.6 File: base.py

class BaseLibSVM(BaseEstimator):
    """Base class for estimators that use libsvm as backing library

    ^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
    ^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

```
jlun@ubuntu: /usr/local/lib/python2.7/dist-packages/sklearn
GNU nano 2.2.6 File: base.py

#####
class BaseEstimator(object):
    """Base class for all estimators in scikit-learn

    ^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
    ^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Implements fit() and predict()



# ML features

- Supervised learning
  - Gen.Lin.Models, Support Vector Machines, Stochastic Gradient Descent, K-NN, Gaussian Processes, Decision Trees, Ensembles (e.g. Random Forests), Feature selection
- Unsupervised learning
  - Gaussian mixture models, Manifold learning, Clustering, Decomposition, Covariance estimation, Outlier detection (explain!), HMM.
- Cross-validation \ Grid search \ Dataset transformation \ Loading data



# Example - Trees and Forests

## A simple decision tree

```
from sklearn.datasets import load_iris

from sklearn import tree

clf = tree.DecisionTreeClassifier()

iris = load_iris()

clf = clf.fit(iris.data, iris.target)

import tempfile

out_file = tree.export_graphviz(clf, "a_simple_tree.dot")

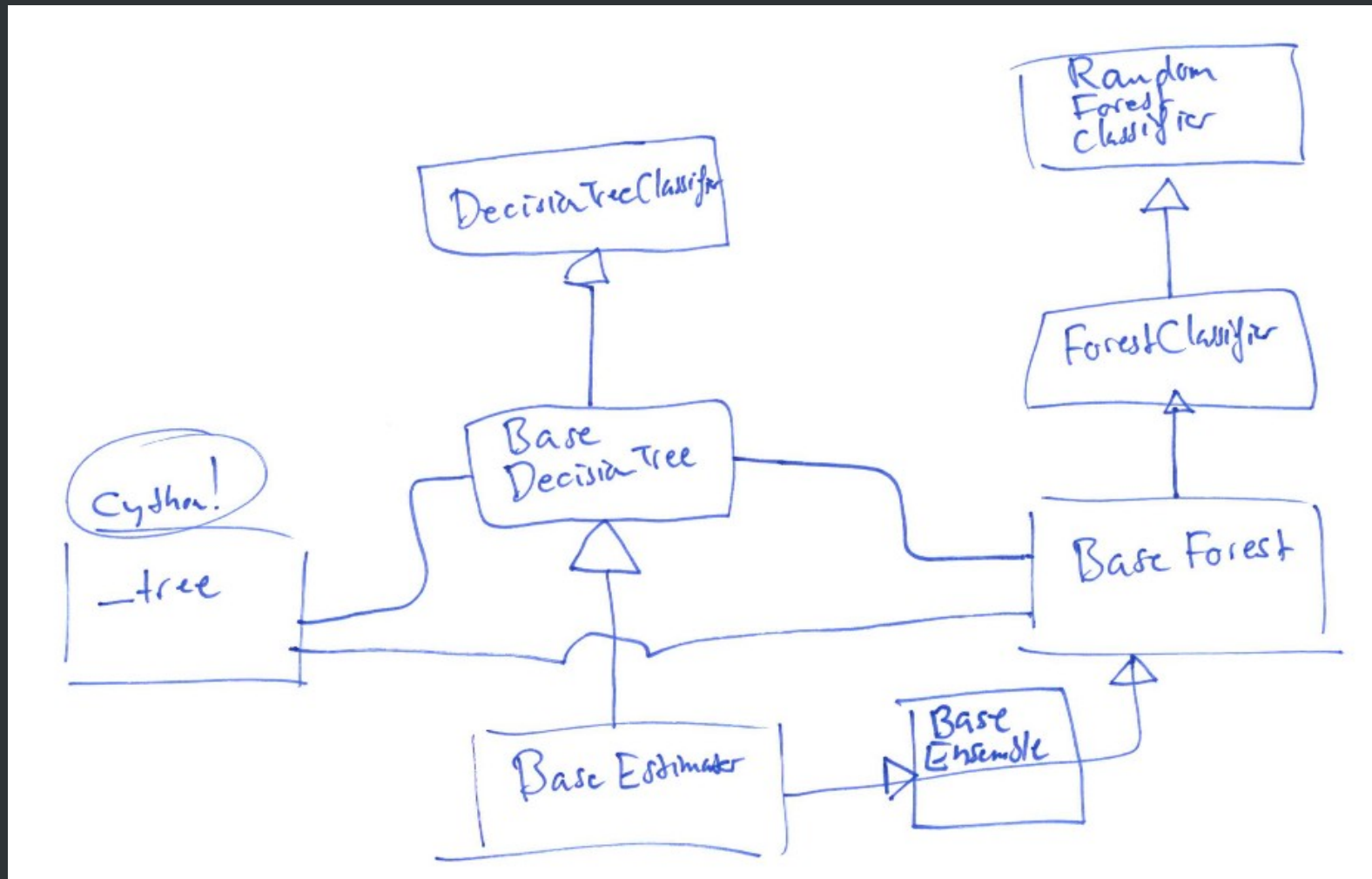
out_file.close()
```

Code demo!



# Example - Trees and Forests

An ensemble of decision trees (RF)



# Pause

10min...



# Hands-on lab: Raspberry Pi Python ML Script

- ARM11 single-core 700MHz with 256MB RAM
- Python language at focus





# Hands-on lab: Raspberry Pi Python ML Script

- *Challenge 1*: Red object recognition on RBP
- *Challenge 2*: Play with toy datasets. Compare ML methods in accuracy, execution speed etc.
- *Challenge 3*: Slawomirs data



# Future work on Scikit-learn (according to authors)

- Online learning
- Scale to large data sets

